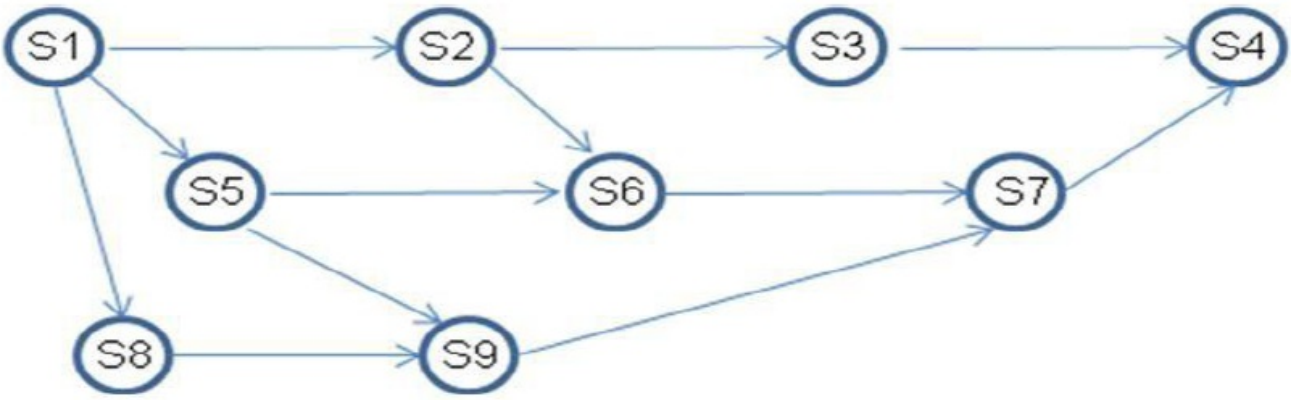
**Shilpita Roy**

# COEN 283 Fall 2015 ASSIGNMENT 2 IPC AND SYNCRONIZATION

**Ques 1: (30 points)**

**Assume you are given the following wait-graph that represents the relationship between multiple threads (s1,s2,s3,...). An arrow from one thread (Sy) to another (Sx) means that thread Sx must finish its computation before Sy starts. (For example: S1 has to wait for S2,S5,S8 to finish, S2 has to wait for s3,s6 to finish and so on.)**
**Use semaphores to enforce this relationship specified by the graph. Be sure to show the initial values and the locations of the semaphore operations. You will be marked based on finding the best solution with minimum number of semaphores.**




Soln: Semaphore a =0, b=0, c=0, d=0, e=0

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| S4<br>Signal(a);<br>Signal(a); | wait(a);<br><br>S3<br>Signal(c); | Wait(a);<br><br>S7<br>Signal(b);<br>Signal(b); | Wait(b);<br><br>S6<br>Signal(c);<br>Signal(d); | Wait(b);<br><br>S9<br>Signal(d);<br>Signal(d); | Wait(c);<br>Wait(c);<br>S2<br>Signal(e); | Wait (d);<br>Wait (d);<br>S5<br>Signal(e); | Wait (d);<br>S8<br>Signal(e); | Wa<br>(e);<br>Wa<br>Wa<br>S1 |

**Ques 2: (10 points)**

**What is the meaning of the term busy waiting?**

Ans : Busy Waiting:

Continuously testing a variable/condition in a process until some value appears or condition becomes true is called busy waiting. The process waits for a lock to be available (variable/ condition) and spins in tight loop till then without relinquishing the CPU. This is called spin lock. This can cause waste of CPU cycles and Priority inversion of processes in Multiprocessing, Non- preemptive Systems.

**What other kinds of waiting are there in an operating system?**

Alternatively, a process can wait for the variable and condition to be true, relinquishing the CPU. It blocks itself on the condition and goes to sleep. When another process sets the condition to true it awakens the sleeping process. This process uses Sleep() and Awake().

**Can busy waiting be avoided altogether? Explain your answer.**

Busy waiting can be avoided but incurs the overhead associated with putting a process to sleep and having to wake it up when the appropriate program state is reached. We can avoid busy waiting by using Synchronization primitives like MUTEX, SEMAPHORE and MONITOR.

**Ques 3: (point 30)**

**The following pair of processes share a common variable X:**

| | Process A | Process B |
|---|---|---|
| **L1:** | int Y | int Z |
| **L2:** | Y = X*2 | Z = X+1 |
| **L3:** | X = Y | X = Z |

**X is set to 5 before either process begins execution. As usual, statements within a process are executed sequentially, but since no assumptions can be made regarding each process's speed of execution, statements in either process may execute in any order with respect to statements in the other process.**
**(a) How many different values of X are possible after both processes finish executing?**

**Ans : Assuming Y and Z are initialized to 0.**
Since there is no Semaphore or lock used the process A and B can be interleaved in many ways before Y or Z is assigned to X.

| Sno | Case | X |
|---|---|---|
| 1 | A: L1->L2-> L3<br>B: L1->L2-> L3 | 11 |
| 2 | A:L1 ->L2<br>B:L1->L2->L3<br>A: L3 | 10 |
| 3 | B:L1->L2<br>A:L1->L2<br>B:L3<br>A:L3 | 10 |
| 4 | A:L1 ->L2<br>B:L1->L2<br>A:L3<br>B:L3 | 6 |
| 5 | B:L1->L2 | 6 |

| | | |
|---|---|---|
| | **A: L1->L2->L3<br>B:L3** | |
| **6** | **B:L1->L2->L3<br>A:L1->L2->L3** | **12** |

**(b) Suppose the programs are modified as follows to use a shared binary semaphore S:**

| | Process A | Process B |
|---|---|---|
| L1: | int Y | int Z |
| L2: | Wait(S) | Wait(S) |
| L3: | Y = X*2 | Z = X+1 |
| L4: | X = Y | X = Z |
| L5: | Signal(S) | Signal(S) |

**S is set to 1 before either process begins execution or, as before, X is set to 5. Now, how many different values of X are possible after both processes finish executing?**
**Ans Assuming Y and Z initialized to 0**
Since there is a Semaphore S then only one process can run while the other waits for the singal and there are only 2 ways the process can be interleaved.

| Sno | Case | X |
|---|---|---|
| **1** | **A: L1->L2->L3->L4->L5<br>B: L1->L2->L3->L4->L5** | **11** |
| **2** | **B: L1->L2->L3->L4->L5<br>A: L1->L2->L3->L4->L5** | **12** |

**(c) Finally, suppose the programs are modified as follows to use a shared binary semaphore T:**

| | Process A | Process B |
|---|---|---|
| L1: | int Y | int Z |
| L2: | Y = X*2 | Wait(T) |
| L3: | X = Y | Z = X+1 |
| L4: | Signal(T) | X = Z |

**T is set to 0 before either process begins execution or, as before, X is set to 5. Now, how many different values of X are possible after both processes finish executing?**
**Ans: The Semaphore T ensures that B always has to wait for A to complete . Thus there is only 1 way for the execution of the two processes**

| | | |
|---|---|---|

| Sno | Case | X |
|---|---|---|
| 1 | A: L1->L2->L3->L4->L5<br>B: L1->L2->L3->L4->L5 | 11 |

## Programming Question

### Ques 4: (30 points)

**Solve the dining philosopher's problem using monitors instead of semaphores**.

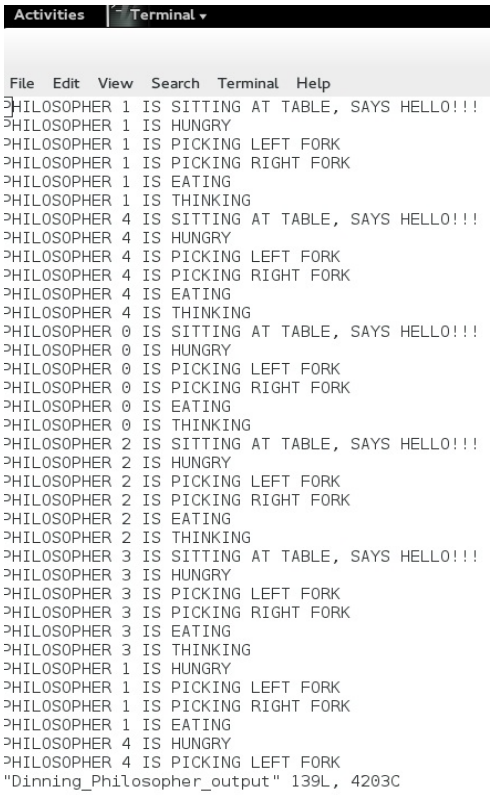
Source code:

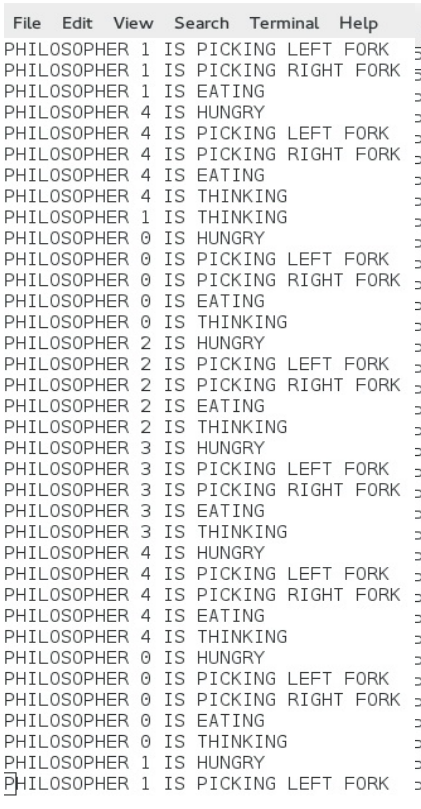
```
/*Dinning p_thread_id using Monitor*/
#include<stdio.h>
#include<pthread.h>
#include<string.h>

extern int random_utime(void);

pthread_mutex_t mutex= PTHREAD_MUTEX_INITIALIZER;

#define NUM_PHILOSOPHER  5

pthread_cond_t status; /* cv */

int activity[NUM_PHILOSOPHER];//0-think  1- hungry  2- eat
int activity_counter_eat[NUM_PHILOSOPHER];

#define TRACE(msg,val)  printf("PHILOSOPHER %d IS %s\n",val,msg) //("%s :%d \n", msg,val)

enum {
          THINK,
          HUNGRY,
          EAT
} state_t;

void pick_fork(int* id)
{
  pthread_mutex_lock(&mutex);

  activity[*id] = HUNGRY;
  TRACE("HUNGRY",*id);
  if(activity[(*id+1)%NUM_PHILOSOPHER]!= EAT && activity[(*id+NUM_PHILOSOPHER-1)%NUM_PHILOSOPHER]!=EAT) {
       activity [*id]=EAT;

       TRACE("PICKING LEFT FORK",*id);
       TRACE("PICKING RIGHT FORK",*id);
      TRACE("EATING",*id);
       activity_counter_eat[*id]++;
  }
  else {
        TRACE("WAITING",*id);
       pthread_cond_wait(&status,&mutex);
  }
  pthread_mutex_unlock(&mutex);
}

void put_fork(int* id)
{
  pthread_mutex_lock(&mutex);
```

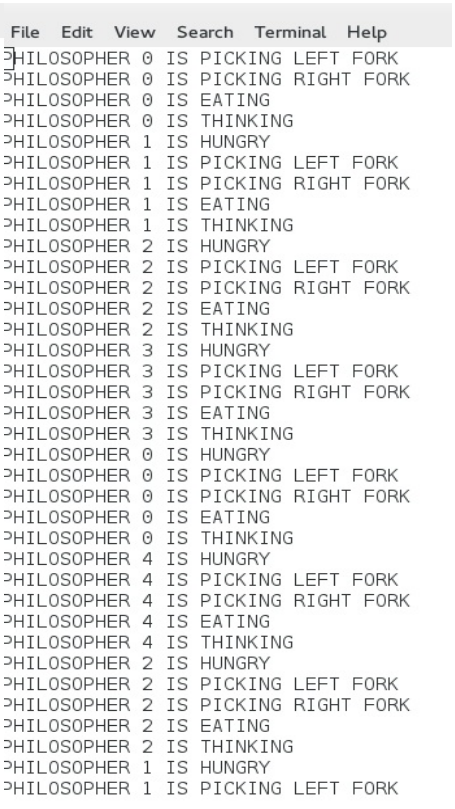
```
  activity[*id]=THINK;
    TRACE("THINKING",*id);
  pthread_cond_signal(&status);
  pthread_mutex_unlock(&mutex);
}

void* dine(void* id)
{
    int* i;
    i =(int*) id;

  //  printf("Philospher %d sits at table\n",*i);
    TRACE("SITTING AT TABLE, SAYS HELLO!!!",*i);

            while(1)
            {
//            TRACE("PHILOSPHER", *i);
                do {
                    if(activity_counter_eat[*i]==5)
                            goto Exit;

                      pick_fork(i);
                } while (activity[*i] == HUNGRY);

                put_fork(i);
                  usleep(random_utime());
            }
    Exit:
    TRACE("LEAVING THE TABLE,SAYS BYE!!!",*i);
    pthread_exit(NULL);
}
/*
void* show_counter(void* arg)
{
       int i=0;
            while(1){
            for( i=0;i<NUM_PHILOSOPHER;i++){
                      printf("%d\t", activity_counter_eat[i]);
                }
                printf("\n");
        }
    pthread_exit(NULL);
}

*/

int main ()
{
  int rc =0;
  int j =0;
  int ret_val=0;
  int arg=0;

  pthread_t p_thread_id[NUM_PHILOSOPHER];
            int eat_stats_id[NUM_PHILOSOPHER];
        int           p_id[NUM_PHILOSOPHER];

//  pthread_t p_thread_observer;

  for(j=0;j< NUM_PHILOSOPHER;j++) {
        p_id[j]=j;
        rc = pthread_create(&p_thread_id[j], NULL, &dine, &p_id[j]);
         if(rc!=0)
                printf("Error Thread Creation");
```

```
 }
 /* ret_val=pthread_create(&p_thread_observer,NULL,&show_counter,&arg);
  if(ret_val!=0)
            printf("Observer Thread creation error");
*/
  for(j=0;j<NUM_PHILOSOPHER;j++) {
                pthread_join(p_thread_id[jj],NULL);
      }
   //    pthread_join(p_thread_observer,NULL);
 pthread_exit(NULL);
 return (0);
}
```

Output shots:



```
File  Edit  View  Search  Terminal  Help
PHILOSOPHER 0 IS PICKING RIGHT FORK
PHILOSOPHER 0 IS EATING
PHILOSOPHER 0 IS THINKING
PHILOSOPHER 4 IS HUNGRY
PHILOSOPHER 4 IS PICKING LEFT FORK
PHILOSOPHER 4 IS PICKING RIGHT FORK
PHILOSOPHER 4 IS EATING
PHILOSOPHER 4 IS THINKING
PHILOSOPHER 2 IS HUNGRY
PHILOSOPHER 2 IS PICKING LEFT FORK
PHILOSOPHER 2 IS PICKING RIGHT FORK
PHILOSOPHER 2 IS EATING
PHILOSOPHER 2 IS THINKING
PHILOSOPHER 1 IS HUNGRY
PHILOSOPHER 1 IS PICKING LEFT FORK
PHILOSOPHER 1 IS PICKING RIGHT FORK
PHILOSOPHER 1 IS EATING
PHILOSOPHER 1 IS THINKING
PHILOSOPHER 3 IS HUNGRY
PHILOSOPHER 3 IS PICKING LEFT FORK
PHILOSOPHER 3 IS PICKING RIGHT FORK
PHILOSOPHER 3 IS EATING
PHILOSOPHER 3 IS THINKING
PHILOSOPHER 0 IS HUNGRY
PHILOSOPHER 0 IS PICKING LEFT FORK
PHILOSOPHER 0 IS PICKING RIGHT FORK
PHILOSOPHER 0 IS EATING
PHILOSOPHER 0 IS THINKING
PHILOSOPHER 2 IS HUNGRY
PHILOSOPHER 2 IS PICKING LEFT FORK
PHILOSOPHER 2 IS PICKING RIGHT FORK
PHILOSOPHER 2 IS EATING
PHILOSOPHER 2 IS THINKING
PHILOSOPHER 4 IS HUNGRY
PHILOSOPHER 4 IS PICKING LEFT FORK
PHILOSOPHER 4 IS PICKING RIGHT FORK
```

```
File  Edit  View  Search  Terminal  Help
PHILOSOPHER 3 IS EATING
PHILOSOPHER 3 IS THINKING
PHILOSOPHER 0 IS HUNGRY
PHILOSOPHER 0 IS PICKING LEFT FORK
PHILOSOPHER 0 IS PICKING RIGHT FORK
PHILOSOPHER 0 IS EATING
PHILOSOPHER 0 IS THINKING
PHILOSOPHER 2 IS HUNGRY
PHILOSOPHER 2 IS PICKING LEFT FORK
PHILOSOPHER 2 IS PICKING RIGHT FORK
PHILOSOPHER 2 IS EATING
PHILOSOPHER 2 IS THINKING
PHILOSOPHER 4 IS HUNGRY
PHILOSOPHER 4 IS PICKING LEFT FORK
PHILOSOPHER 4 IS PICKING RIGHT FORK
PHILOSOPHER 4 IS EATING
PHILOSOPHER 3 IS HUNGRY
PHILOSOPHER 3 IS WAITING
PHILOSOPHER 1 IS HUNGRY
PHILOSOPHER 1 IS PICKING LEFT FORK
PHILOSOPHER 1 IS PICKING RIGHT FORK
PHILOSOPHER 1 IS EATING
PHILOSOPHER 1 IS THINKING
PHILOSOPHER 3 IS HUNGRY
PHILOSOPHER 3 IS WAITING
PHILOSOPHER 4 IS THINKING
PHILOSOPHER 3 IS HUNGRY
PHILOSOPHER 3 IS PICKING LEFT FORK
PHILOSOPHER 3 IS PICKING RIGHT FORK
PHILOSOPHER 3 IS EATING
PHILOSOPHER 3 IS THINKING
PHILOSOPHER 0 IS LEAVING THE TABLE,SAYS BYE!!!
PHILOSOPHER 2 IS LEAVING THE TABLE,SAYS BYE!!!
PHILOSOPHER 1 IS LEAVING THE TABLE,SAYS BYE!!!
PHILOSOPHER 4 IS LEAVING THE TABLE,SAYS BYE!!!
PHILOSOPHER 3 IS LEAVING THE TABLE,SAYS BYE!!!
```